```
/***************************************************************************
*************************  Appendix A     **********************************
************************  Least Square Lattice  ****************************
***********************    Noise Cancelling     ****************************/
/* Example for ratiometric approach to noise cancelling */
#define LAMBDA  0.95

void OxiLSL_NC( int     reset,
                int     passes,
                int     *signal_1,
                int     *signal_2,
                int     *signal_3,
                int     *target_1,
                int     *target_2) {

        int     i, ii, k, m, n, contraction;
static  int     *s_a, *s_b, *s_c, *out_a, *out_c;
static  float   Delta_sqr, scale, noise_ref;


 if( reset == TRUE){
   s_a    = signal_1;
   s_b    = signal_2;
   s_c    = signal_3;
   out_a  = target_1;
   out_c  = target_2;
   factor = 1.5;
   scale  = 1.0 /4160.0;

 /* noise canceller initialization at time t=0 */

  nc[0].berr  = 0.0;
  nc[0].Gamma = 1.0;

  for(m=0; m<NC_CELLS; m++) {
    nc[m].err_a   = 0.0;
    nc[m].err_b   = 0.0;
    nc[m].Roh_a   = 0.0;
    nc[m].Roh_c   = 0.0;
    nc[m].Delta   = 0.0;
    nc[m].Fswsqr  = 0.00001;
    nc[m].Bswsqr  = 0.00001;
  }
 }

/*======================== END INITIALIZATION ===========================*/

 for(k=0; k<passes; k++){

   contraction = FALSE;
   for(m=0; m< NC_CELLS; m++) {        /* Update delay elements        */
     nc[m].berr1   =  nc[m].berr;
     nc[m].Bswsqr1 =  nc[m].Bswsqr;
   }


  noise_ref   = factor * log(1.0 - (*s_a) * scale)
              - log(1.0 - (*s_b) * scale) ;
  nc[0].err_a = log(1.0 - (*s_b) * scale);
  nc[0].err_b = log(1.0 - (*s_c) * scale);
```

```
++s_a;
++s_b;
++s_c;

nc[0].ferr   = noise_ref ;
nc[0].berr   = noise_ref ;
nc[0].Fswsqr = LAMBDA * nc[0].Fswsqr + noise_ref * noise_ref;
nc[0].Bswsqr = nc[0].Fswsqr;

/* Order Update      */
for(n=1;( n < NC_CELLS) && (contraction == FALSE); n++) {

  /* Adaptive Lattice Section */

  m = n-1;
  ii= n-1;

  nc[m].Delta *=  LAMBDA;
  nc[m].Delta +=  nc[m].berr1 * nc[m].ferr  / nc[m].Gamma ;
  Delta_sqr    =  nc[m].Delta * nc[m].Delta;

  nc[n].fref   = -nc[m].Delta / nc[m].Bswsqr1;
  nc[n].bref   = -nc[m].Delta / nc[m].Fswsqr;

  nc[n].ferr   =  nc[m].ferr  + nc[n].fref * nc[m].berr1;
  nc[n].berr   =  nc[m].berr1 + nc[n].bref * nc[m].ferr;

  nc[n].Fswsqr =  nc[m].Fswsqr  - Delta_sqr / nc[m].Bswsqr1;
  nc[n].Bswsqr =  nc[m].Bswsqr1 - Delta_sqr / nc[m].Fswsqr;

  if( (nc[n].Fswsqr + nc[n].Bswsqr) > 0.00001 || (n < 5) ) {
    nc[n].Gamma = nc[m].Gamma - nc[m].berr1 * nc[m].berr1 / nc[m].Bswsqr1;
    if(nc[n].Gamma < 0.05) nc[n].Gamma = 0.05;
    if(nc[n].Gamma > 1.00) nc[n].Gamma = 1.00;

  /* Joint Process Estimation Section */

    nc[m].Roh_a *= LAMBDA;
    nc[m].Roh_a += nc[m].berr * nc[m].err_a  / nc[m].Gamma ;
    nc[m].k_a    = nc[m].Roh_a  / nc[m].Bswsqr;
    nc[n].err_a  = nc[m].err_a  - nc[m].k_a * nc[m].berr;

    nc[m].Roh_c *= LAMBDA;
    nc[m].Roh_c += nc[m].berr * nc[m].err_b  / nc[m].Gamma ;
    nc[m].k_c    = nc[m].Roh_c  / nc[m].Bswsqr;
    nc[n].err_b  = nc[m].err_b  - nc[m].k_c * nc[m].berr;

  }
  else {
    contraction = TRUE;
    for(i=n; i<NC_CELLS; i++) {
      nc[i].err_a   = 0.0;
      nc[i].Roh_a   = 0.0;
      nc[i].err_b   = 0.0;
      nc[i].Roh_c   = 0.0;
      nc[i].Delta   = 0.0;
      nc[i].Fswsqr  = 0.00001;
      nc[i].Bswsqr  = 0.00001;
      nc[i].Bswsqr1 = 0.00001;
```

```
      }

    }

  }

  *out_a++ = (int)( (-exp(nc[ii].err_a) +1.0) / scale) ;
  *out_c++ = (int)( (-exp(nc[ii].err_b) +1.0) / scale) ;


 }
}
/***********************    Least Square Lattice    *****************************
************************                           ******************************
*******************************************************************************/
```